A General and Parallel Platform for Mining Co-Movement Patterns over Large-scale Trajectories

ABSTRACT

1. INTRODUCTION

The prevalence of positioning devices has drastically boosted the scale and spectrum of trajectory collection to an unprecedented level. Tremendous amounts of trajectories, in the form of sequenced spatial-temporal records, are continually generated from animal telemetry chips, vehicle GPSs and wearable devices. Data analysis on large-scale trajectories benefits a wide range of applications and services, including traffic planning [1], animal analysis [2], and social recommendations [3], to name just a few.

A crucial task of data analysis on top of trajectories is to discover co-moving patterns. A *co-movement* pattern [4] refers to a group of objects traveling together for a certain period of time and the group is normally determined by spatial proximity. A pattern is prominent if the size of the group exceeds M and the length of the duration exceeds K, where M and K are parameters specified by users. Rooted from such basic definition and driven by different mining applications, there are a bunch of variants of co-movement patterns that have been developed with more advanced constraints.

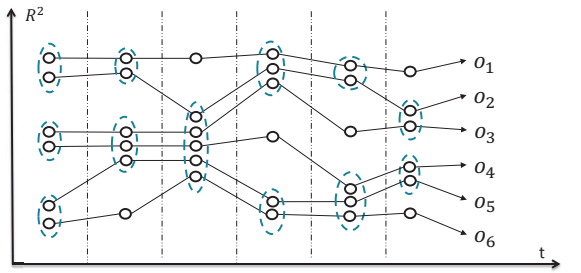
Table 1 summarizes several popular co-moving pattern s with different constraints in the attributes of clustering in spatial proximity, consecutiveness in temporal duration and computational complexity. In particular, the *flock* [5] and the *group* [6] patterns require all the objects in a group to be enclosed by a disk with radius r; whereas the *convoy* [7], the *swarm* [8] and the *platoon* [9] patterns resort to density-based spatial clustering. In the temporal dimension, the *flock* [5] and the *convoy* [7] require all the timestamps of each detected spatial group to be consecutive, which is referred to as *global consecutiveness*; whereas the *swarm* [8] does not impose any restriction. The *group* [6] and the *platoon* [9] adopt a compromised manner by allowing arbitrary gaps between the consecutive segments, which is called *local consecutiveness*. They introduce a parameter L to control the minimum length of each local consecutive segment.

Patterns	Proximity	Consecutiveness	Time Complexity
flock [10]	disk-based	global	$O(\|\mathbb{O}\|\|\mathbb{T}\|(M + log(\|\mathbb{O}\|))$
convoy [7]	density-based	global	$O(\|\mathbb{O}\|^2 + \|\mathbb{O}\|\|\mathbb{T}\|)$
swarm [8]	density-based	-	$O(2^{\|\mathbb{O}\|}\|\mathbb{O}\|\|\mathbb{T}\|)$
group [6]	disk-based	local	$O(\|\mathbb{O}\|^2\|\mathbb{T}\|)$
platoon [9]	density-based	local	$O(2^{\|\mathbb{O}\|}\|\mathbb{O}\|\|\mathbb{T}\|)$

Table 1: Constraints and complexity of co-movement patterns. The time complexity indicates the performance in the worst case, where $|\mathbb{O}|$ is the total number of objects and $|\mathbb{T}|$ is the number of descritized timestamps.

Figure 1 is an example to demonstrate the concepts of various co-movement patterns. The trajectory database consists of six moving objects and the temporal dimension is discretized into six snapshots. In each snapshot, we treat the clustering methods as a black-box and assume that they generate the same clusters. Objects in proximity are grouped in the dotted circles. As aforementioned, there are three parameters to determine the co-movement patterns and the default settings in this example are $M = 2$, $K = 3$ and $L = 2$. Both the *flock* and the *convoy* require the spatial clusters to last for at least K consecutive timestamps. Hence,$\{o_3, o_4\}$ and $\{o_5, o_6\}$ remains the only two candidates matching the patterns. The *swarm* relaxes the pattern matching by discarding the temporal consecutiveness constraint. Thus, it generates many more candidates than the *flock* and the *convoy*. The *group* and the *platoon* add another constraint on local consecutiveness to retain meaningful patterns. For instance, $\{o_1, o_2 : 1, 2, 4, 5\}$ is a pattern matching local consecutiveness because timestamps $\{1, 2\}$ and $\{4, 5\}$ are two segments with length no smaller than $L = 2$. The difference between the *group* and the *platoon* is that the *platoon* has an additional parameter K to specify the minimum number of snapshots for the spatial clusters. This explains why $\{o_3, o_4, o_5 : 2, 3\}$ is a *group* pattern but not a *platoon* pattern.

As can be seen, there are various co-movement patterns requested by different applications and it is cumbersome to design a tailored solution for each type. In addition, despite the generality of the *platoon* (i.e., it can be reduced to other types of patterns via proper parameter settings), it suffers from the so-called *loose-connection* anomaly. We use two objects o_1 and o_2 in Figure 2 as an example to illustrate the scenario. These two objects form a *platoon* pattern in timestamps $\{1, 2, 3, 102, 103, 104\}$. However, the two consecutive segments are 98 timestamps apart, resulting in a false positive co-movement pattern. In reality, such an anomaly may be caused by the periodic movements of unrelated ob-



Flock(M=2, K=3)	$\{o_3, o_4: 1,2,3\}$ $\{o_5, o_6: 3,4,5\}$
Convoy(M=2, K=3)	$\{o_3, o_4: 1,2,3\}$ $\{o_5, o_6: 3,4,5\}$
Group(M=2, L=2)	$\{o_1, o_2: 1,2,4,5\}$ $\{o_3, o_4, o_5: 2,3\}$ $\{o_5, o_6: 3,4,5\}$
Platoon (M=2,K=3,L=2)	$\{o_1, o_2: 1,2,4,5\}$ $\{o_3, o_4: 1,2,3\}$ $\{o_4, o_5: 2,3,5,6\}$ $\{o_5, o_6: 3,4,5\}$
Swarm (M=2,K=3)	$\{o_1, o_2: 1,2,4,5\}$ $\{o_3, o_4: 1,2,3\}$ $\{o_4, o_5: 2,3,5,6\}$ $\{o_5, o_6: 1,3,4,5\}$ $\{o_2, o_3: 3,4,6\}$

Figure 1: Trajectories and co-movement patterns; The example consists of six trajectories across six snapshots. Objects in spatial clusters are enclosed by dotted circles. M is the minimum cluster cardinality; K denotes the minimum number of snapshots for the occurrence of a spatial cluster; and L denotes the minimum length for local consecutiveness.

jects, such as vehicles stopping at the same petrol station or animals pausing at the same water source. Unfortunately, none of the existing patterns have directly addressed this anomaly.

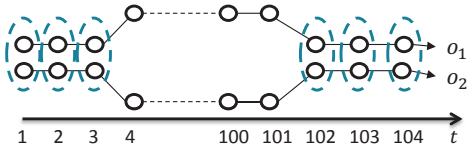
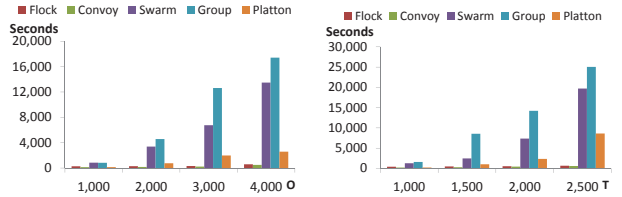


Figure 2: *Loose-connection* anomaly. Even though $\{o_1, o_2 : 1, 2, 3, 102, 103, 104\}$ is considered as a valid *platoon* pattern, it is highly probable that these two objects are not related as the two consecutive segments are 98 timestamps apart.

The other issue with existing methods is that they are built on top of centralized indexes which may not be scalable. Table 1 shows their theoretical complexities in the worst cases and the largest real dataset ever evaluated in previous studies is up to million-scale points collected from hundreds of moving objects. In practice, the dataset is of much higher scale and the scalability of existing methods is left unknown. Thus, we conduct an experimental evaluation with 4000 objects moving for 2500 timestamps to examine the scalability. Results in Figure 3 show that their performances degrade dramatically as the dataset scales up. For instance, the detection time of *group* drops twenty times as the number of objects grows from *1k* to *4k*. Similarly, the performance of *swarm* drops over fifteen times as the number of snapshots grows from *1k* to *2.5k*. These observations imply that existing methods are not scalable to support large-scale trajectory databases.

Therefore, our primary contributions in this paper are to close these two gaps. First, we propose the *general co-movement pattern* (GCMP) which models various co-moment patterns in a unified way and can avoid the *loose-connection*



(a) $1k$ timestamps (b) $1k$ objects

Figure 3: Performance measures on existing co-movement patterns. A sampled Geolife data set is used with up two 2.4 million data points. Default parameters are $M = 10$ $K = 20$ $L = 10$.

anomaly. In GCMP, we introduce a new gap parameter G to pose a constraint on the temporal gap between two consecutive segments. By setting a feasible G, the loose-connection anomaly can be avoided. In addition, our GCMP is both general and expressive. It can be reduced to any of the previous patterns by customizing the parameters.

Second, we investigate deploying our GCMP detector on MapReduce platforms (such as Hadoop and Spark) to tackle the scalability issue. Our technical contributions are threefold. First, we replicate the snapshots in multiple data chunks to support efficient parallel processing. Second, we devise a novel *Star Partition and Mining* (SPM) algorithm as a fine-granularity partitioning strategy to achieve workload balance. For each star, an Apriori method is adopted to mine the co-movement patterns. Third, we propose two types of optimization techniques to further improve the performance, including *edge-simplification* to boost the shuffle process and *temporal monotonicity pruning* and *forward closure checking* to significantly reduce the number of enumerated candidates in the Aproiro algorithm.

We conduct a set of extensive experiments on XXX datasets with billion-scale trajectory points. The results show that XXX.

The rest of our paper is organized as follows: Section 2 summarizes the relevant literature on trajectory pattern mining. Section 3 states the problem definition of our general co-movement pattern mining. Section 4 provides a baseline *temporal replication and mining* solution. An advanced solution named *star partition and mining* is presented in Section 5. Section 6 discusses various optimization techniques. Section 7 conducts extensive experiments to verify the usefulness and efficiency of our system. Finally Section 8 concludes the paper.

2. RELATED WORKS

The *co-movement patterns* in literature consist of five members, namely *group* [6], *flock* [10], *convoy* [7], *swarm* [8] and *platoon* [9]. We have demonstrated the semantics of these patterns in Table 1 and Figure 1. In this section, we focus on comparing the techniques used in these works. For more trajectory patterns other than *co-movement patterns*, interested readers may move to [11] for a comprehensive survey.

2.1 Flock and Convoy

The difference between *flock* and *convoy* lies in the object clustering methods. In *flock* objects are clustered based on their distance. Specifically, the objects in the same cluster

needs to have a pair-wised distance less than *min_dist*. This essentially requires the objects to be within a disk-region of delimiter less than *min_dist*. In contrast, *convoy* cluster the objects using density-based clustering [12]. Technically, *flock* utilizes a m^{th}-order Voronoi diagram [13] to detect whether a subset of object with size greater than m stays in a disk-region. *Convoy* employs a trajectory simplification [14] technique to boost pairwise distance computations in the density-based clustering. After clustering, both *flock* and *convoy* use a line-sweep method to scan each snapshots. During the scan, the object group appears in consecutive timestamps is detected. Meanwhile, the object groups that do not match the consecutive constraint are pruned. However, such a method faces high complexity issues when supporting other patterns. For instance, in *swarm*, the candidate set during the line-sweep grows exponentially, and many candidates can only be pruned after the entire snapshots are scanned.

2.2 Group, Swarm and Platoon

Different from *flock* and *convoy*, all the *group,swarm* and *platoon* patterns have more constraints on the pattern duration. Therefore, their techniques of mining are of the same skeleton. The main idea of mining is to grow object set from an empty set in a depth-first manner. During the growth, various pruning techniques are provided to prune unnecessary branches. *Group* pattern uses the Apriori property among patterns to facilitate the pruning. *Swarm* adapts two more pruning rules called backward pruning and forward pruning. *Platoon* further adapts a prefix table structure to guide the depth-first search. As shown by Li et.al. [9], *platoon* outperforms other two methods in efficiency. However, the three patterns are not able to directly discover the general co-movement pattern. Furthermore, their pruning rules heavily rely on the depth-first search nature, which lost its efficiency in the parallel scenario.

2.3 MaReduce Framework

MapReduce (MR) was formally proposed by Dean et.al. [15] and has subsequently implemented by many open source systems. Those systems provide handy APIs with fault tolerances and are popularly used as large-scale data processing platforms. In simple words, there are two conceptual types of computing nodes in MR, namely the *mapper*s and the *reducer*s. The execution of a MR algorithm consists of three major steps: First, input data are partitioned and read by a *map* function on each mapper. Then, mappers emit key-value pairs which are *shuffle*d over the network to reducers. Lastly, reducers process the received data using a *reduce* function.

3. DEFINITIONS

Let $\mathbb{O} = \{o_1, o_2, ..., o_n\}$ be the set of objects and $\mathbb{T} = \{1, 2, ..., m\}$ be the descritized temporal dimension. A time sequence T is defined as a subset of $\mathbb{T}$, i.e., $T \subseteq \mathbb{T}$, and we use $|T|$ to denote sequence length. Let T_i be i-th entry in T and we say T is consecutive if $\forall 1 \leq i \leq |T|-1, T_{i+1} = T_i + 1$. It is obvious that any time sequence T can be decomposed into consecutive segments and we say T is *L-consecutive* [9] if the length of all the consecutive segments is no smaller than L.

As illustrate in Figure 2, patterns adapting the notion of L-consecutiveness (e.g., *platoon* and *group*) still suffer from *loose connection* problem. To avoid such an anomaly without losing pattern generality, we introduce a parameter G to control the gaps between timestamps in a pattern. Formally, a G-connected time sequence is defined as follows:

Pattern	M	K	L	G	Clustering
Group	2	1	2	$\|\mathbb{T}\|$	Disk-based
Flock	·	·	K	1	Disk-based
Convoy	·	·	K	1	Density-based
Swarm	·	·	1	$\|\mathbb{T}\|$	Density-based
Platoon	·	·	·	$\|\mathbb{T}\|$	Density-based

Table 2: Expressing other patterns using GCMP. · indicate a user specified value. M represents the object *size* constraints. K represents *duration* constraint. L represents *consecutiveness* constraint. G represents the *connection* constraints.

Definition 1 (G-connected)**.** *A time sequence T is G-connected if the gap between any of its neighboring timestamps is no greater than G. That is $\forall T_i, T_{i+1} \in T, T_{i+1} - T_i \leq G$.*

We take $T = \{1, 2, 3, 5, 6\}$ as an example, which can be decomposed into two consecutive segments $\{1, 2, 3\}$ and $\{5, 6\}$. T is not 3-consecutive since the length $\{5, 6\}$ is 2. Thus, it is safe to say either T is 1-consecutive or 2-consecutive. On the other hand, T is 2-connected since the maximum gap between its neighboring time stamps is $5 - 3 = 2$. It is worth noting that T is not 1-connected because the gap between T_3 and T_4 is 2 (i.e., 5-3=2).

Given a trajectory database descritized into snapshots, we can conduct a clustering method, either disk-based or density-based, to identify groups with spatial proximity. Let T be the set of timestamps in which a group of objects O are clustered. We are ready to define a more general co-movement pattern:

Definition 2 (General Co-Movement Pattern)**.** *A general co-movement pattern finds a set of objects O satisfying the following five constraints: 1) closeness: the objects in O belong to the same cluster in the timestamps of T; 2) significance: $|O| \geq M$; 3) duration: $|T| \geq K$; 4) consecutiveness: T is L-consecutive; and 5) connection: T is G-connected.*

There are four parameters in our general co-movement pattern, including object constraint M and temporal constraints K, L, G. By customizing these parameters, our pattern can express other patterns proposed in previous literature, as illustrated in Table 2. In particular, by setting $G = |T|$, we achieve the *platoon* pattern. By setting $G = |T|, L = 1$, we achieve the *swarm* pattern. By setting $G = |T|$, $M = 2$, $K = 1$, we gain the *group* pattern. Finally by setting $G = 1$, we achieve the *convoy* and *flock* pattern. In addition to the flexibility of representing other existing patterns, our GCMP is able to avoid the *loose connection* anomaly by tuning the parameter G. It is notable that GCMP cannot be modeled by existing patterns.

It is also observable that the number of patterns in GCMP could be exponential under some parameter settings (i.e., when expressing *swarm*). In particular, given a parameter M, if a pattern P is valid, then any subset of P with size M is also a valid pattern. This results in additional $\Sigma_{M \geq i \geq |P.O|} \binom{|P.O|}{i}$ patterns, which is clearly overwhelming and redundant. For all these patterns, output P is sufficient.

Therefore, we define the *Closed General Co-Movement Pattern* as follows:

Definition 3 (Closed General Co-Movement Pattern)**.** *A general co-moving pattern $P = \langle O : T \rangle$ is closed if and only if there does not exist another general co-moving pattern P' s.t. $P.O \subseteq P'.O$.*

For example, let $M = 2, K = 2, , L = 1, G = 1$. In Figure 1, the pattern $P_1 = \{o_3, o_4 : 1, 2, 3\}$ is not a closed pattern. This is because $P_2 = \{o_3, o_4, o_5 : 2, 3\}$ is a closed pattern since $P_2.O \supset P_1.O$. The closed pattern avoids outputting duplicate information, thus making the result patterns more compact.

Our definition of GCMP is free from clustering method. Users are able to supply different clustering methods to facilitate different application needs. We currently expose both disk-region based clustering and DBSCAN as options to the user.

In summary, the goal of this paper is to present a parallel solution for discovering closed GCMP from large-scale trajectory data.

Before we move on to the algorithmic part, we list the notations that are used in the following sections.

Symbols	Meanings
S_t	snapshot of objects at time t
$\mathbb{O}$	set of objects
M	object size constraint
K	duration constraint
L	consecutiveness constraint
G	connection constraint
$P = (O : T)$	pattern with object set O, time sequence T
η	replication factor in TRM
$C_t(o)$	the cluster of object o at time t
S_t	the set of clusters at time t
λ_i	the partition with snapshots $S_t, .., S_{t+\eta-1}$
Sr_i	the star of object i

Table 3: Symbols and notions that will be used

4. BASELINE: TEMPORAL REPLICATION AND PARALLEL MINING

We resort to MapReduce (MR) as a general, parallel and scalable paradigm for GCMP pattern mining. Since a GCMP pattern may cross multiple temporal segments, we cannot simply split the trajectory database into disjoint partitions and conduct parallel mining within each partition. Instead, we need certain data replication to guarantee that no matching patterns are missing.

We propose a framework named *Temporal Replication and ParallelMining* (TRPM), as illustrated in Figure 4. The mining procedure consists of two cycles of map-reduce jobs connected in a pipeline manner. In the first map-reduce cycle, input trajectories are grouped into snapshots and then clustered. In particular, object locations at the same timestamps form a snapshot in the map-phase. In the reduce phase, objects in a snapshot are clustered by the supplied clustering methods (e.g., DBSCAN or disk-based) in parallel. The reducers finally output clusters of objects in each snapshot, represented by a list of $\langle t, S_t \rangle$ where t is the timestamp and S_t is a set of clustered object at snapshot t. The outputs will be fed into the second cycle of map-reduce jobs, whose details are presented in Algorithm 1. Intuitively, we first replicate each snapshot and each consecutive segment with η snapshots is considered as one data partition. Then, a *Line Sweeping* method is developed to discover GCMP within each partition in the reduce stage. The final patterns from all partitions are then stored into external storage (e.g, HDFS). In the following, we are interested to examine the minimum η and introduce the *Line Sweeping* algorithm in details.

Algorithm 1 Temporal Replication and Parallel Mining

Require: list of $\langle t, S_t \rangle$ pairs
1: $\eta \leftarrow (\lceil \frac{K}{L} \rceil - 1) * (G - 1) + 2K - 2$
2: —Map Phase—
3: **for all** $\langle t, S_t \rangle$ **do**
4: **for all** $i \in 1...\eta - 1$ **do**
5: emit a $\langle \max(t - i, 0), S_t \rangle$ pair
6: **end for**
7: **end for**
8: —Partition and Shuffle Phase—
9: **for all** $\langle t, S \rangle$ pair **do**
10: group-by t, emit a $\langle t, \lambda_t \rangle$
11: where $\lambda_t = \{S_t, S_{t+1}, ..S_{t+\eta-1}\}$
12: **end for**
13: —Reduce Phase—
14: **for all** $\langle t, \lambda_t \rangle$ **do**
15: lineSweepMining(λ_t)
16: **end for**

In temporal replication, every partition contains η consecutive snapshots. Therefore, determining η is critical in ensuring the correctness and efficiency of TRPM. If η is too small, some cross-partition patterns may be missed out. On the other hand, if η is too large, the shuffle and reduce stage of TRPM would incur expensive costs. To guarantee the *complete* discovery of all patterns, we enforce the following statement on η: *Let T' be the* shortest *valid subsequence (wrt. K, L, G) of a valid temporal sequence T. Then, η needs to be the upper bound for all T' among all valid temporal sequences to ensure the completeness.* Finding the *minimum* η is not straightforward. We notice the minimum η is related to the temporal parameters K, L, G by making the following observation on $G = 1$:

Let T be an arbitrary valid temporal sequence. Since G equals to 1, T can be represented as $T = (t_1, t_2, ..., t_m)$, where $t_i + 1 = t_{i+1}$ and $m \geq K$. Let $T' = (t_1, ..., t_k)$, T' is the shortest valid temporal sequence of T. Note that T' is fully contained in the partition $\lambda_{t_1} = \{S_{t_1}, ..., S_{t_k}\}$, therefore it can be mined in λ_{t_1}. Since T is an arbitrary valid temporal sequence, the minimum η is thus K.

Inspired by the observation, we generalize the relationship between the minimum η and temporal parameters using the following theorem:

Theorem 1. *The minimum η to ensure the completeness of the temporal replication is:*

$$\eta = \begin{cases} (\lceil \frac{K}{L} \rceil - 1) * (G - 1) + 2K - 2, & \text{if } G \geq 2 \\ K, & \text{if } G = 1 \end{cases}$$

, where K, L, G are the temporal parameters.

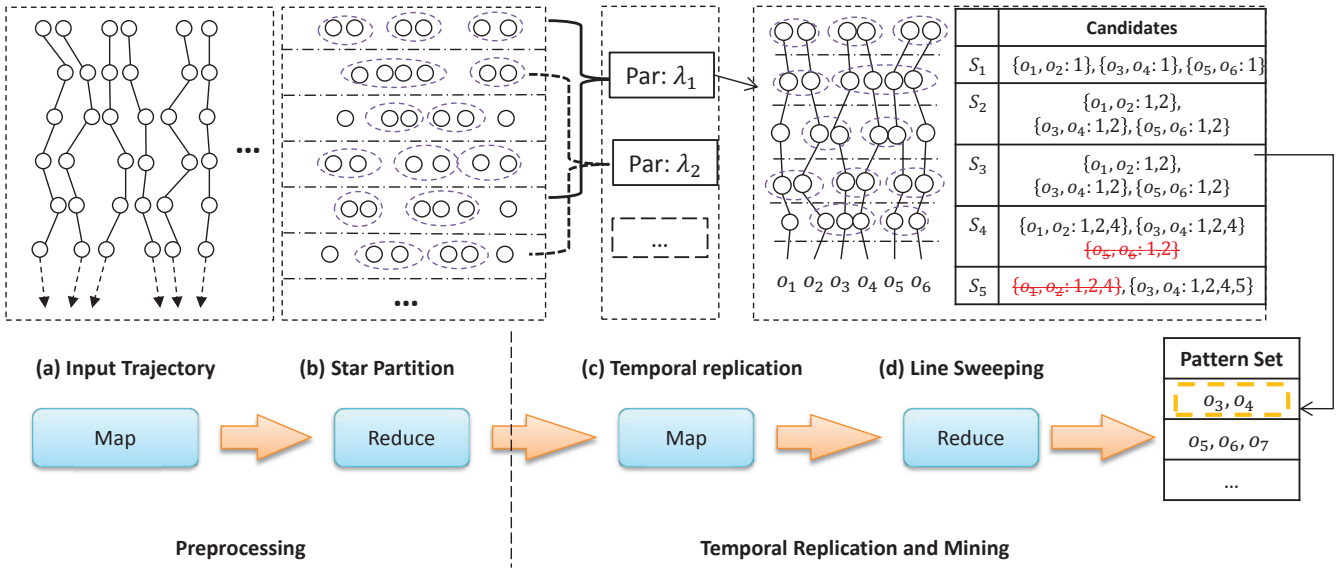


Figure 4: Work flow of Temporal Replication and Mining. (a)(b) correspond to the first map-reduce cycle which clusters objects in each snapshot; (c)(d) correspond to the second map-reduce cycle which uses temporal replication to mine GCMP in parallel.

Proof. When $G = 1$, we have demonstrated $\eta = K$ in the above observation. When $G \geq 2$, we compute η as follows: any T' can be viewed as n consecutive segments with sizes $l_1, .., l_n$ and $n-1$ gaps with sizes $g_1, ..., g_{n-1}$. Since η is the upper bond among all T's, η can be formulated as follows:

$$\eta = \max_{n, l_i, g_i} \{\Sigma_{i=1}^{i=n} l_i + \Sigma_{i=1}^{i=n-1} g_i\} \quad (1)$$

With the following constraints: (1)$\forall l_i, L \leq l_i \leq K-1$; (2) $\forall g_i, 1 \leq g_i \leq G-1$; (3) $\Sigma_{i=1}^{i=n} l_i \geq K$ and (4) $\Sigma_{i=1}^{i=n-1} l_i \leq K-1$. Constraints (1)(2)(3) due to the validity of T' and $G \geq 2$ (4) is because $|T'|$ is minimum. Based on (1)(3)(4), we can derive (5): $n \in [1, \lceil \frac{K}{L} \rceil]$. Constraints (1)-(5) form a convex polygon and η is monotone increasing wrt. n, l_i, g_i, therefore, the maximum value of η is taken at the upper boundaries where $g_i = G-1$, $\Sigma_{i=1}^{i=n-1} l_i = K-1$, $l_i = K-1$ and $n = \lceil \frac{K}{L} \rceil$. This leads to $\eta = (\lceil \frac{K}{L} \rceil - 1) * (G-1) + 2K - 2$. □

Based on the above theorem, during TRPM, every consecutive η snapshots form a partition. In particular, for every snapshot S_t, there is a partition $\lambda_t = \{S_t, ..., S_{t+\eta-1}\}$. With this partition strategy, when discovering GCMPS in λ_t, we only need to keep the pattern candidates whose object set are in S_t. This motivates to design an efficient *line-sweep* mining (LSM) method for each partition. The intuition of LSM is to sequentially scan all snapshots. During scanning, a set of pattern candidates is maintained. When all snapshots are scanned, the remaining candidates are the true patterns. The detail of LSM is presented in Algorithm 2. A candidate set C is maintained throughout the algorithm(line 1). C is initialized by inserting clusters at S_t (lines 2-4). During scanning snapshot S_j, candidates in C are joined with clusters at S_j. In the join, a candidate grows its temporal sequence while potentially reduces its object set. After the join, false patterns are deleted (line 7). Note that the size of C is always decreasing, therefore the complexity of LSM λ_t is $O(\eta |S_t| |\overline{S}|)$,where $|\overline{S}|$ is the average snapshot size in λ_t.

Algorithm 2 Line Sweep Mining

Require: $\lambda_t = \{S_t, ..., S_{t+\eta-1}\}$
1: $C \leftarrow \{\}$ ▷ Candidate set
2: **for** $c \in S_t$ **do**
3: C.add($\langle c, t \rangle$)
4: **end for**
5: **for all** $j \in [t+1, t+\eta-1]$ **do**
6: $C \leftarrow S_j \oplus C$
7: remove false candidates from C
8: **end for**
9: output true candidates in C

Example 1. *We illustrate the work flow of TRPM method using Figure 4 (c)(d) with pattern parameters $M = 2, K = 3, L = 2, G = 2$. By Theorem 1, η is calculated as $(\lceil \frac{K}{L} \rceil - 1) * (G-1) + 2K - 2 = 5$. Therefore, in Figure 4 (c), every 5 consecutive snapshots are combined into a partition in the map phase. In Figure 4 (d), a line sweep method is illustrated for partition λ_1. Let C_i be the candidate set during sweeping snapshot S_i. Initially, C_1 contains patterns whose object set is in snapshot S_1. As line sweeps, the patterns in C_i grows. At snapshot S_4, the candidate $\{o_5, o_6\}$ is removed. This is because the gap between its latest timestamp (i.e., 2) and the next scanning timestamp (i.e., 5) is 3, which violates the G constraint. Next, at snapshot S_5, the candidate $\{o_1, o_2\}$ is removed. This is because its local consecutive timestamps $\{4\}$ has only size 1, which violates the L constraint. Finally, $\{o_3, o_4\}$ is the qualified pattern and is outputted. Note that the minimum η under this setting is 5. If η is chosen as 4, the pattern $\{o_3, o_4\}$ would be excluded.*

Although TRPM works in a parallel way, we observe two inefficiencies. First, the performance of TRPM largely relies

on η which further depends on pattern parameters. When η is large, the shuffle and reduce cost of TRPM is high. Second, the parallel execution of line sweep algorithm may discover the same pattern from multiple partitions, which introduces redundant work. To resolve the two limitations, we design a novel *Star Partition and Mining* algorithm.

5. STAR PARTITION AND MINING

To resolve the two limitations of TRM, we propose the *Star Partition and Mining* (SPM) method. Instead of partition trajectory database in the temporal domain, SPM partition the trajectory database in the object domain. Specifically, we design a graph model, named *connection graph*, to capture the co-moving behavior among objects. In such a way, given an object u, all the objects potentially co-moved with u are captured in u's neighborhood, which forms a *star* structure in graph terms. SPM utilizes such structures to partition objects and then adapts an Apriori algorithm to discover true GCMPs from each partition.

The flow of SPM method is presented in Figure 5. SPM utilizes the same preprocessing as TRM, thus the flow starts from clusters in each snapshot. Conceptually, the clusters in each snapshot forms a graph as shown in Figure 5 (a). In the map phase of *SPM*, a vertex together with its neighborhood vertexes form a star. Every star is indeed an independent partition. Then, stars are shuffled to reducers. As in Figure 5 (c), Apriori mining algorithm is adapted to mine the GCMP patterns. The overview implementation of SPM is shown in Algorithm 3.

Algorithm 3 Star Partition and Mining

Require: list of $\langle t, S_t \rangle$ pairs
1: —Map phase—
2: **for all** $C \in S_t$ **do**
3: **for all** $(o_1, o_2) \in C \times C$ **do**
4: **if** $o_1 < o_2$ **then**
5: emit a $\langle o_1, o_2, \{t\} \rangle$ triplet
6: **end if**
7: **end for**
8: **end for**
9: —Partition and Shuffle phase—
10: **for all** $\langle o_1, o_2, \{t\} \rangle$ triplets **do**
11: group-by o_1, emit $\langle o_1, Sr_{o_1} \rangle$
12: **end for**
13: —Reduce phase—
14: **for all** $\langle o, Sr_o \rangle$ **do**
15: Apriori(Sr_o)
16: **end for**

5.1 Star Partition

The intuition of the star partition is that, if two objects are part of the same pattern, they must belong to the same cluster at those snapshots. Therefore, we may link objects that belong to the same cluster to form the *connection graph*. Objects that are not connected surely fail to form a pattern. We may then partition the connection graph based on vertex connectivity such that mining GCMPs can be done in parallel. We formally define the *connection graph* as follows:

Definition 4 (Connection Graph)**.** *A connection graph is an undirected graph $G = (V : E)$, where each $v \in V$ represents an object. An edge $e(s,t) = ET \in E$ contains the timestamp sequence at which s, t are in the same cluster, i.e., $\forall t \in ET, C_t(s) = C_t(t)$.*

In graph theory, a *star* of a vertex u is the the set of neighborhood vertexes of u. To utilize *star* partition, since a vertex may appear in multiple stars, some replications of vertexes are required. In order to avoid replication of edges, we use the *directed star* as follows:

Definition 5 (Directed Star)**.** *Assign each vertex in G an ID, a direct star of a vertex s, denoted as Sr_s, is the set of incidental edges on s such that $\forall e(s,t) \in Sr_s$, $s < t$. We name s as the* central vertex *of Sr_s.*

By leveraging the *directed star*, we avoids replicating the edges. A *Connection graph* and *star* examples are shown in Figure 5 (a) and (b). In (a), a connection graph is formed based on the example in Figure 1. In (b), 5 stars are presented. It can be figured out that by leveraging directed star, no edges are replicated. In implementation, as show in Algorithm 3 line 4, the comparison between vertices/objects are based on the vertex/object IDs.

5.2 Apriori Mining

To systematically discover valid patterns in each star, we design the *Apriori Mining* algorithm. To describe the algorithm, we call a candidate pattern R-pattern if the size of its object set is R. Therefore, each edge in the star is effectively a 2-pattern. The intuition of Apriori mining is the observation that $(R+1)$-patterns can be generated from R-patterns and 2 patterns. Thus, we may iteratively enumerate pattern candidates with all possible sizes. In particular, initially, for each $e(s,v) = ET$, pattern $p = (\{s,v\}, ET)$ is formed. During each iteration, we generate $(R+1)$-patterns by joining R-patterns with the 2-patterns. Technically, the join between $p_1 = (O_1 : T_1)$ and $p_2 = (O_2 : T_2)$ generates a new pattern $p_3 = (O_1 \cup O_2 : T_1 \cap T_2)$. Note that in Sr_s, each R-pattern contains the object s, thus the join only grow a R-pattern at most to a $(R+1)$-pattern. Our mining algorithm stops where no further patterns are generated. The algorithm is illustrated as in Algorithm 4.

Algorithm 4 Apriori Mining

Require: Sr_s
1: Lv $\leftarrow \{\}$,Ground $\leftarrow \{\}$, Output $\leftarrow \{\}$
2: **for all** $e(s,t) = T \in Sr_s$ **do**
3: Simply $e(s,t)$ ▷ Edge Simplification
4: Ground.add($\langle \{s,t\}, T \rangle$)
5: Lv $\leftarrow$ Ground
6: **end for**
7: **while** Lv is not empty **do**
8: Lv $\leftarrow$ Lv $\oplus$ Ground
9: Remove false patterns in Lv ▷ Temporal Monotonicity
10: **if** union of Lv is a valid patter **then**
11: output.add(Lv) ▷ Forward Closure
12: break
13: **end if**
14: **end while**
15: output.addAll(Lv)
16: **return** output

An illustration of Algorithm 4 is shown in Figure 5 (c). As shown, the star $Sr_3 = \{3,4,5,6\}$ initially generate three

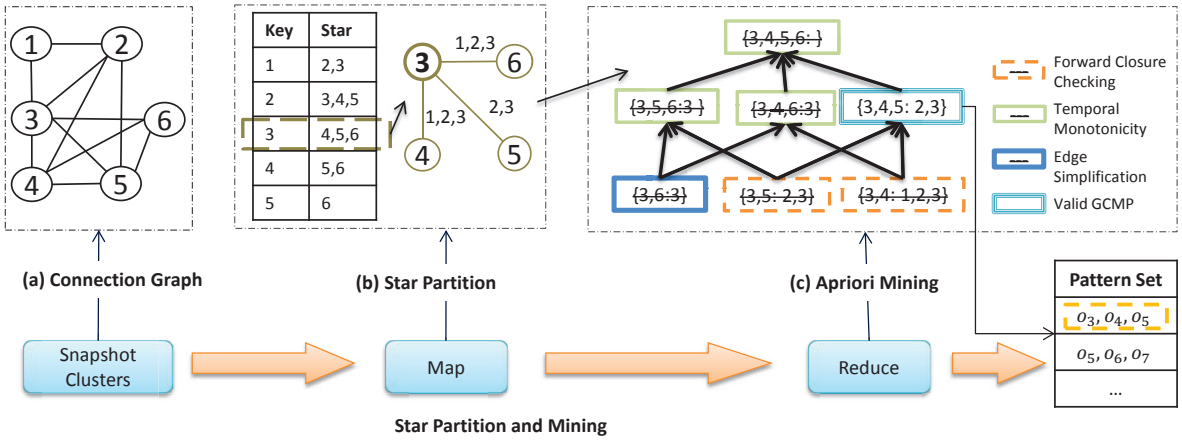


Figure 5: Star partition and mining. (a) Conceptual connection graph from Figure 1.(b) Five star partitions are generated (c) Apriori Mining with various pruning techniques.

2-candidates. At every iteration, higher level candidates are generated by joining lower level candidates. When no more candidates can be generated, the algorithm stops by outputting the valid patterns.

It is notable that, in star partition, original data is replicated at most $O(|\mathbb{O}|)$ times. In later sections, we will describe several more optimization techniques to further reduce the amount of replicated data.

5.3 Analysis of SPM

The analysis of SPM contains two parts. We first prove the correctness of SPM. Then, we analyze the work load distribution of the star sizes.

5.3.1 Correctness

An important concern in star partition is that whether any valid patterns are missed out by such a partition. We assert the correctness by the following theorem:

Theorem 2 (Correctness of Star Partition)**.** *Star partition is complete.*

In fact, besides that every valid patterns can be mined in some stars, star partition further ensures that patterns discovered from different star partitions cannot be the same. This can be formulated as:

Lemma 3 (Unique Patterns in Star Partition)**.** *Let Sr_i and Sr_j $(i \neq j)$ be two stars. Let P_i (resp. P_j) be the patterns discovered from Sr_i (resp. P_j). Then, $\forall p_i \in P_i, \forall p_j \in P_j, p_i.O \not\equiv p_j.O$.*

This lemma indicates that every star partition discovers different patterns, thus less redundant work could exist as opposed to the line sweep method in TRM.

5.3.2 Optimal Star Partition

An important concern in design parallel algorithms is the distribution of work loads. Traditionally, the quality of a partition strategy is measured based on two aspects: (1) the number of result partition, which affects the maximum parallelism (2) the size balance of partitions, which affects the finishing time of a job. Unlike TRM where each partition contains equal-sized snapshots, the size distribution of stars in SPM remains unknown. Nevertheless, we notice that, in SPM, the total sizes of stars are invariant. Therefore, the quality of a star partition can be formalized as the *skewness*, which is the maximum star size among all stars. Smaller *skewness* naturally results in more partitions and less imbalance.

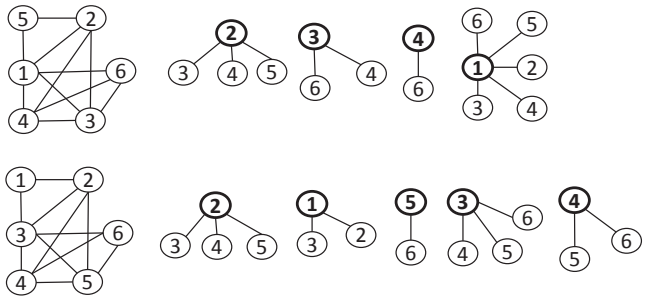


Figure 6: An alternative numbering and partitioning of the connection graph in Figure 5.

Interestingly, we note that the *skewness* of star partition is affected by the way the vertexes are numbered in the connection graph. For example, Figure 6 gives two valid numbering of vertexes of the same connection graph, but produces two different set of stars. The upper partitioning constructs 4 stars with the maximum star consisting of 5 edges. The lower partitioning constructs 5 stars with the maximum star consisting of 3 edges. Apparently the upper partitioning is inferior because its *skewness* is 5 while the lower one's *skewness* is 3.

Ideally we wish to find a numbering scheme of connection graph that minimize the *skewness*. To quantify, we use an linear algebra model as follows: Let G denote a connection graph. Let $\mathbb{A}$ be an arbitrary numbering of vertexes in G. Let $(A : a_{i,j})$ be a boolean assignment matrix wrt. $\mathbb{A}$ (i.e., $a_{i,j}$ indicates whether vertex j is included in Sr_i). Let vector $\vec{b}$ be the *one*[1] vector. Let $\vec{c} = A\vec{b}$, then each $c_i \in \vec{c}$ denotes the size of the star Sr_i. Therefore, minimizing the *skewness*

[1]Every element in $\vec{b}$ is 1

can be formulated as follows:

$$\mathbb{A} = \text{argmin}(||A\vec{b}||_\infty), \text{where } ||A\vec{b}||_\infty = \max_{1 \le j \le n}(c_j) \quad (2)$$

It is challenging to directly optimize the above equation. First, suppose there are n vertexes in G, enumerating all possible $\mathbb{A}$s leads to $n!$ combinations. Such a high complexity is trivially unpractical. Second, since G is only conceptual at runtime, the load planning cannot be done beforehand. Despite these challenges, we observe that there is a $O(1)$ time solution which is good enough as stated in the following theorem.

Theorem 4 (Balance of Star Partition)**.** *Let G be a connection graph with n vertexes and the average degree d. Let $\mathbb{A}^*$ be the optimal numbering wrt. Equation 1. For an random numbering, $\mathbb{A}$, with high probability, the absolute skewness difference between $\mathbb{A}^*$ and $\mathbb{A}$ is $O(\sqrt{n \log n})$. That is, it is with probability $1 - 1/n$, $||\mathbb{A}\vec{b}||_\infty = ||\mathbb{A}^*\vec{b}||_\infty + O(\sqrt{n \log n})$.*

In fact, we have a tighter bound of $||A\vec{b}||_\infty - ||A^*\vec{b}||_\infty$ if the connection graph is *dense*. Specifically, if $d \ge \sqrt{12 \log n}$, the following equation holds:

$$Pr(||A\vec{b}||_\infty \ge (d/2 + O(\sqrt{d \log n})) \le 1/n$$

Utilizing Theorem 3, in SPM, it is safe to choose the object IDs as the vertex number. This because in reality, object IDs are often hashed and can be deemed as random.

6. OPTIMIZING SPM

In Algorithm 4, there are two major factors affecting the performance. First, the edges in Sr_s affects the initial size of 2-patterns. Second, the candidates generated in each level affects the join performance. In this section, we describe several practical techniques to boost the performance.

6.1 Edge Simplification

Each edge $e(s, v)$ in Sr_s contains a time sequence ET which represents the co-occurrence of s and v. We notice that the edge between s and v is not always necessary. For example, if an edge has a cardinality less than K, it is unnecessary to include this edge to Sr_s since it cannot contribute to any patterns. This motivates us to simplify the edges in Sr_s.

Our intuition is thus to, remove the unnecessary timestamps among the edges in a star, such that the valid patterns would not be affected. To realize this idea, we formally design the concept of *candidate sequence* as follows:

Definition 6 (Candidate Sequence)**.** *Given the pattern parameters: L, K, G, a sequence T is a* Candidate Sequence *if for any of its maximal G-connected sequence T', T' conforms to L, K.*

For example, let $L = 2, K = 4, G = 2$, sequence $T_1 = (1, 2, 4, 5, 6, 9, 10, 11, 13)$ is not a candidate sequence since one of its maximal 2-connected sequence $(9, 10, 11, 13)$ does not conform to L. In contrast, sequence $T_2 = (1, 2, 4, 5, 6, 8, 9)$ is a candidate sequence.

Not that, although T_1 is not a candidate sequence, it can be simplified to a candidate sequence (i.e, T_2). Generally, for any time sequence T, if it does not contain a *candidate sequence*, then it cannot contribute to any valid patterns. In fact, only the candidate sequence inside T affects the final patterns. Therefore, we may use the candidate sequence to simplify edges in a star. During the simplification, if an edge results in a candidate sequence of size zero, then it is removed from the star. Otherwise, the simplified edge is kept. By so doing, the size of the new star is much smaller.

We design an efficient linear time algorithm for the simplification process. It takes two rounds scan of T as shown in Algorithm 5. In the first round, the consecutive portions of T with size less than L are removed. In the second round, the maximal G-connected sequences of size less than K are removed. Clearly the simplification algorithm takes $O(T)$ time.

Algorithm 5 Edge Simplification

```
Require: T
—L Phase—
c ← 0
for i ∈ (0, ..., |T|) do
    if T[i] − T[i − 1] > 1 then
        if i − c < L then
            T remove [c : i)
        end if
        c ← i
    end if
end for
—G-K Phase—
s ← 1, c ← 0
for i ∈ (0 : |T|) do
    if T[i] − T[i − 1] > G then
        if s < K then
            T remove [c : i)
        end if
        c ← i, s ← 1
    else
        s + +
    end if
end for
```

Example 2. *Take $T_1 = \{1, 2, 4, 5, 6, 9, 10, 11, 13\}$ as an example of edge simplification. Let $L = 2, K = 4, G = 2$. In the first round of scan. T_1 reduces to $\{1, 2, 4, 5, 6, 9, 10, 11\}$. The consecutive subsequence $\{13\}$ is removed by $L = 2$. T_1 has two maximal 2-consecutive subsequences, which are $\{1, 2, 4, 5, 6\}$ and $\{9, 10, 11\}$. Since $K = 4$, $\{9, 10, 11\}$ is removed from T_1 in the second round of scan. Therefore, T_1 is simplified to $\{1, 2, 4, 5, 6\}$, which is shortened 44%.*

6.2 Temporal Monotonicity

In Apriori, we repeatedly join candidate patterns with different sizes. We observe that traditional monotonic property of Apriori algorithms **does not** hold in GCMP mining. Specifically, given two candidates P_1, P_2, if $P_1.O \subset P_2.O$ and P_1 is not a valid pattern, then P_2 may or may not be a valid pattern. However, we notice that we may form another monotonic property based on the *candidate sequence*.

The intuition is that if a candidate $P_1.T$ cannot be simplified to a *candidate sequence*, then P_1 cannot be valid pattern. Furthermore, any candidate P_2, with $P_1.O \subset P_2.O$ cannot be a valid pattern. This *temporal monotonic property* is explicitly described as in the follow theorem:

Theorem 5 (Temporal Monotonicity)**.** *Given the temporal parameters L, G, K, for a candidate pattern P, if $P.T$ cannot*

be simplified to a candidate sequence, then, for any candidate P' with $P.O \subseteq P'.O$, P' cannot be a true pattern.

6.3 Forward closure checking

Although leveraging *temporal monotonicity* could largely prune false candidates and reduce the apriori search space, it is ineffective when a *true* pattern exists. In an extreme case, if a final pattern of a star Sr_s is the *union of all vertices* in the star, then in apriori, $\binom{|Sr_s|}{i+1}$ candidates needs to be generated at each level i. This results in an exponential search space while the output only contains one pattern. Generally, when candidates at level i collectively forms a true pattern, running aprior produces many wasted candidates.

Let Lv_i be the set of candidates at level i of Algorithm 4, we use the *forward closure* FC_i to denote the union of the objects in all candidates in Lv_i. Then, the *forward closure checking* is stated as follows:

Theorem 6 (Forward Closure Checking)**.** *Let Lv_i be the candidates generated at level i in Algorithm 4, if FC_i is a proper pattern, then it is safe to terminate Algorithm 4 and directly output FC_i.*

It is notable that, as the level grows in Algorithm 4, the closure FC reduces, thus the pruning power of FC would be stronger. We give hybrid example of the three optimization as follows:

Example 3. *We use Figure 5 (c) to demonstrate the power of candidate pruning. As shown, at the initial stage, $\{3, 6 : 3\}$ is first pruned by Edge Simplification since its timestamps fails to be a candidate sequence. Subsequently, all further candidates containing $\{3, 6\}$ are pruned by Temporal Monotonicity. Then, we check the Forward Closure of remaining candidates (i.e., $\{3, 4\}$ and $\{3, 5\}$) and find $\{3, 4, 5\}$ is a valid candidate. Therefore, $\{3, 4\}$ and $\{3, 5\}$ are pruned, and $\{3, 4, 5\}$ is the output.*

7. EXPERIMENTAL STUDY

7.1 Experimental Setup

We adapt one of the most popular MapReduce platform, Apache Spark, to conduct experiments on mining GCMPs. Our experiments run on a 9-node cluster, with Apache Yarn as the cluster manager. We use 1 node for Yarn resource manager, and use the remaining 8 nodes as executors. Each node in the cluster is uniformly equipped with a 2.2GHz quad-core CPU with 32 GB memory. Inter-node communication is carried by the 1Gbps Ethernet. Some critical configuration of Spark is as follows:

Parameter	Value
Java Version	1.7.0
spark.driver.memory	2GB
spark.executor.cores	2
spark.executor.instances	11
spark.executor.memory	7GB
spark.master	yarn-cluster
spark.serializer	KryoSerializer

We prepare three real datasets for study. The details of the datasets are as follows:

- Geolife [2]: contains 18,670 GPS trajectories for passengers in Beijing over three years. The data are collected per 1 5 seconds. We use linear interpretation for missing data in short-intervals (i.e., 5 seconds), since interpolating big intervals introduces large errors.
- ACTShopping [3]: contains visitors trajectories in ATC shopping center in Osaka from October 24, 2012 to November 29, 2013. There are in total 18670 trajectories, with 41 million data points.
- SingTaxi: contains 15,054 Singapore taxi trajectories in August 2012.

The summary of statistics are presented as in the following table:

Table 4: Statistics of data set

	Geolife	ACTShopping	SingTaxi
# objects	18,670	13,183	15,054
# average ts	2,924	3,114	19,667
# longest ts	17,281	24,407	43,717
# data points	54,594,696	41,052,242	296,075,837

7.2 Effects of G on GCMP

We study the effects of the gap parameter G on discovering co-movement patterns. As shown in Figure 3, existing works are not able to handle large trajectories, we sample and cluster 1 million trajectory points from the three real data. We use two instances of GCMPs, GCMP-L where the L value is default and GCMP where $L = 1$. We measure the number of patterns discvoered by different methods. The results are presented in Figure 7.

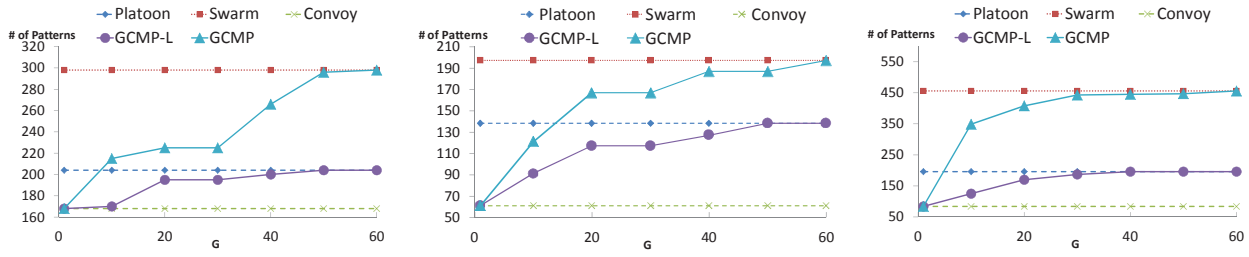
We can see from Figures 7 that the number of patterns discovered by GCMP is a positively related to G. When $G = 1$, the number of patterns returned by GCMP and GCMP-L are identical to *convoy*. As G grows, the number returned by GCMP-L converges to *platoon*. For example, in (a), two patterns converge when G is 40. Similarly in (b) and (c), two patterns converge when G is 50. The reason is that when G is large enough, all consecutive segments in *platoon* satisfy the G-connectivity. Similar trend applies between GCMP and *swarm*. As see from the figures, in (a), two patterns converge at $G = 50$; while in (b) and (c) the two patterns converge at $G = 60$ and $G = 40$ respectively. We further note that, existing patterns has different granularity in temporal domain. As we seen from the three figures, the number of *swarm* is greater than *platoon* and *convoy*. Despite none of the existing patterns are able to fine-grain control the temporal domain of patterns, GCMP can discover the patterns more precisely. These results confirms the usefulness of GCMP in expressing other patterns.

7.3 Performance Comparison

We now study the performance of our GCMP methods. We use TRPM as the baseline algorithm and implement SPM and SPM+.

7.3.1 Effects of clusters ϵ, minPt

[2]http://research.microsoft.com/en-us/projects/geolife/
[3]http://www.irc.atr.jp/crest2010_HRI/ATC_dataset/



(a) Geolife (b) ACTShopping (c) SingTaxi

Figure 7: Effectiveness of GCMP model on sampled datasets.

Variables	Meaning	Values
M	min size of object set	20, 40, **60**,80,100
K	min duration	60, 120, **180**, 240, 300
L	min local duration	10, 20, **30**, 40,50
G	max gap	5,10,**15**,20,25
ϵ	DBSCAN parameter	3,6,**9**,12
minPt	DBSCAN parameter	3,6,**9**,12

Table 5: Pattern Parameters and their default values.

7.3.2 *Effects of pattern parameters* M, L, K, G

7.4 SPM Analysis

7.4.1 *Load Balance*

7.4.2 *Scalability*

8. CONCLUSION AND FUTURE WORK

9. REFERENCES

[1] Y. Zheng, Y. Liu, J. Yuan, and X. Xie, "Urban computing with taxicabs," in *Proceedings of the 13th international conference on Ubiquitous computing*, pp. 89–98, ACM, 2011.

[2] Z. Li, B. Ding, J. Han, R. Kays, and P. Nye, "Mining periodic behaviors for moving objects," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1099–1108, ACM, 2010.

[3] J. Bao, Y. Zheng, D. Wilkie, and M. F. Mokbel, "A survey on recommendations in locationbased social networks. submitted to," *Geoinformatica*, 2013.

[4] X. Li, *Managing moving objects and their trajectories*. PhD thesis, National University of Singapore, 2013.

[5] J. Gudmundsson and M. van Kreveld, "Computing longest duration flocks in trajectory data," in *Proceedings of the 14th annual ACM international symposium on Advances in geographic information systems*, pp. 35–42, ACM, 2006.

[6] Y. Wang, E.-P. Lim, and S.-Y. Hwang, "Efficient mining of group patterns from user movement data," *Data & Knowledge Engineering*, vol. 57, no. 3, pp. 240–282, 2006.

[7] H. Jeung, M. L. Yiu, X. Zhou, C. S. Jensen, and H. T. Shen, "Discovery of convoys in trajectory databases," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 1068–1080, 2008.

[8] Z. Li, B. Ding, J. Han, and R. Kays, "Swarm: Mining relaxed temporal moving object clusters," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 723–734, 2010.

[9] Y. Li, J. Bailey, and L. Kulik, "Efficient mining of platoon patterns in trajectory databases," *Data & Knowledge Engineering*, 2015.

[10] J. Gudmundsson, M. van Kreveld, and B. Speckmann, "Efficient detection of motion patterns in spatio-temporal data sets," in *Proceedings of the 12th annual ACM international workshop on Geographic information systems*, pp. 250–257, ACM, 2004.

[11] Y. Zheng, "Trajectory data mining: an overview," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 6, no. 3, p. 29, 2015.

[12] D. Birant and A. Kut, "St-dbscan: An algorithm for clustering spatial–temporal data," *Data & Knowledge Engineering*, vol. 60, no. 1, pp. 208–221, 2007.

[13] P. Laube, M. van Kreveld, and S. Imfeld, "Finding remodetecting relative motion patterns in geospatial lifelines," in *Developments in spatial data handling*, pp. 201–215, Springer, 2005.

[14] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *Cartographica: The International Journal for Geographic Information and Geovisualization*, vol. 10, no. 2, pp. 112–122, 1973. doi:10.3138/FM57-6770-U75U-7727.

[15] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

APPENDIX

A. PROOFS OF THEOREMS

A.1 Proof of Theorem 1

A.2 Proof of Theorem 2 and Lemma 3

Proof. If P is a valid pattern in original trajectories, let s be the object with smallest ID in $P.O$. Based on the definition of GCMP, $\forall t \in P.T$, $\forall o \in P.O$, $C_t(s) = C_t(o)$. It follows that all object $o \in P.O$ are in Sr_s. Furthermore, every timestamp in $P.T$ is included in Sr_s. Therefore, P is a valid pattern in Sr_s. This completes the proof of Theorem 2. The correctness of Lemma 3 can be directly derived from the above arguments: let o_i be the smallest element in $p_i.O$ and o_j be the smallest element in $p_j.O$. Since p_i and p_j belongs to different stars, then $o_i \neq o_j$, therefore $p_i.O \not\equiv p_j.O$. This proves the lemma. □

A.3 Proof of Theorem 4

Proof. Let $\mathbb{A}^*$ be the optimal numbering wrt Equation 1. Since we have a star for each object, by the degree-sum formula and pigeon-hole theorem, $||A^*\vec{b}||_\infty \geq d/2$. Next, let $e_{i,j}$ be an entry in the adjacent matrix of G. Note that edges in G are independent. Let d_i denote the degree of vertex i in G. It follows that $E[d_i] = E[\Sigma_{1\leq j\leq n} e_{i,j}] = d$. Since in the star partition, each edge is

assigned to the vertex with smaller IDs, the connection between $a_{i,j}$ and $e_{i,j}$ can be written as:

$$a_{i,j} = \begin{cases} e_{i,j}, i > j \\ 0, otherwise \end{cases}$$

There are two observations made on the above equation. First, since $e_{i,j}$s are independent, $a_{i,j}$s are independent. Second, since $i > j$ and $e_{i,j}$ are independent. $E[a_{i,j}] = E[e_{i,j}]E[i > j] = E[e_{i,j}]/2$.

By definition, $c_i = \Sigma_{1 \leq j \leq n} a_{i,j}$, is a sum of n independent 0-1 variables. Taking expectation on both sides, we get: $E[c_i] = E[\Sigma_{1 \leq j \leq n} a_{i,j}] = E[\Sigma_{1 \leq j \leq n} e_{i,j}]/2 = d/2$. Let $\mu = E[c_i] = d/2$, $t = \sqrt{n \log n}$, by Hoeffding's Inequality, the following holds:

$$\begin{aligned} Pr(c_i \geq \mu + t) &\leq \exp(\frac{-2t^2}{n}) \\ &= \exp(-2 \log n) = n^{-2} \end{aligned}$$

The first step is due to the fact that all $a_{i,j}$ are bounded in the range of [0,1]. Next, since the event $(\max_{1 \leq j \leq n}(c_j) \geq \mu + t)$ can be viewed as $\cup_{c_i}(c_i \geq \mu + t)$, by Union Bound, we achieve the following:

$$\begin{aligned} Pr(||A\vec{b}||_\infty \geq \mu + t) &= Pr(\max_{1 \leq j \leq n}(c_j) \geq \mu + t) \\ &= Pr(\cup_{c_i}(c_i \geq \mu + t)) \\ &\leq \Sigma_{1 \leq i \leq n} Pr(c_i \geq \mu + t) \\ &= n^{-1} = 1/n \end{aligned}$$

Substitute back t and μ, we achieve the following concise form:

$$Pr(||A\vec{b}||_\infty \geq (d/2 + \sqrt{n \log n})) \leq 1/n$$

This indicates that, the probability of $(||A\vec{b}||_\infty - d/2)$ being less than or equal to $O(\sqrt{n \log n})$ is $(1 - 1/n)$. With the observed fact that $||A^*\vec{b}||_\infty \geq d/2$, we conclude that with probability greater than $(1 - 1/n)$, the difference between $||A\vec{b}||_\infty$ and $||A^*\vec{b}||_\infty$ is less than $O(\sqrt{n \log n})$. □

A.4 Proof of Theorem 5

Proof. Let P_1, P_2 be two candidates with $P_1.O \subseteq P_2.O$. It is easy to see that $P_1.T \supseteq P_2.T$. Suppose $P_1.T$ cannot be simplified to a candidate sequence. Then by proof of contradiction, any subset of $P_1.T$ cannot be simplified. It follows that $P_2.T$ cannot be simplified to a candidate sequence. In summary, if $P_1.T$ cannot be simplified, P_2 can be pruned. □

A.5 Proof of Theorem 6

Proof. We prove by contradiction. Suppose there exists another pattern P such that $P.O \neq FC_i$, let $X = P.O - FC_i \neq \emptyset$. Consider a subset P_1 of P which contains X with size $i + 1$, (i.e., $P_1 \subseteq P, P_1 \subseteq X, |P_1| = i + 1$). Since P is a proper pattern, then P_1 is also a proper pattern. Therefore $P_1 \in Lv_i$. It follows X is in the forward closure of FC_i, (i.e.,$X \in FC_i$), which contradicts with $X \notin FC_i$. □